Moby: Collaborative Editing in Desktop Applications

Pavel Curtis

Collaboratively constructed work products

- Presentations
- Documents
- Project schedules
- Source code
- CAD drawings
- Network diagrams
- etc.

Current collaboration via email

- Everybody writes their own section separately, in their own style / formatting
- Everybody sends their section to one owner / victim
- Victim merges sections together, unifying differing styles / formatting
- Victim sends result out to everyone
- Victim acts as central clearinghouse for all future edits, goes quietly insane

What do I mean by "collaborative editing"?

- Multiple users modifying multiple copies of the same data model
  - Each user modifies only their local copy
- Information about modifications is communicated between the users
- Each copy eventually reflects the effects of all of the users' modifications

What do users need?

- Freedom
  - Users are never prevented from editing
    - Whether others are editing or not
    - Users can edit while disconnected
- Convenience
  - Users don't have to perform tedious merges
    - But will be alerted when auto-merges threaten intent
- Control
  - Owner can review others' edits and selectively reject them
- Flexible deployment
  - No hard requirement for a central server

Requirements for useful collaborative editing

- Convergence
  - All copies eventually reach the same state
  - Can be satisfied trivially
    - E.g., delete all content everywhere!
- Intention preservation
  - All edits have exact effect intended by user
  - Obviously, can only infer/preserve low-level intentions
    - E.g., delete the right word, insert in the right place
  - Critical for asynchronous document co-editing

Sending edits, not full models

- Only send descriptions of individual edits:
  - "delete *N* characters starting at position *P*"
  - "insert the text *X* at position *P*"
  - etc.
- Edits are typically much smaller than models
- Inferring edits by comparing models is often inaccurate, arbitrary, and unintuitive

Approaches to supporting collaborative editing

- State-based (à la Word)
- Operation-based
  - Mutual exclusion
    - Locking
    - Transactions
    - Token passing (à la OneNote)
  - Unrestricted simultaneous editing
    - Do / Undo / Redo (à la Groove)
    - Operation Transformation (à la Live Meeting)

Sending edits, not full models

- Only send descriptions of individual edits:
  - "delete *N* characters starting at position *P*"
  - "insert the text *X* at position *P*"
  - etc.
- Edits are typically much smaller than models
- Inferring edits by comparing models is often inaccurate, arbitrary, and unintuitive

Approaches to supporting collaborative editing

- State-based (à la Word)
- Operation-based
  - Mutual exclusion
    - Locking
    - Transactions
    - Token passing (à la OneNote)
  - Unrestricted simultaneous editing
    - Do / Undo / Redo (à la Groove)
    - Operation Transformation (à la Live Meeting)

Locking

- Enforce "at most one writer"
- Broadcast updates from writer when lock is released (or more often)
- Scope of each lock can vary
  - Whole data model (e g document, presentation)
  - Logical pieces (e g , paragraph, slide)
- Can manage locks explicitly or implicitly
  - Explicit: pre-assign document sections to authors
  - Implicit whenever user moves selection

Locking issues

- Works slowly peer-to-peer
  - Requires ackowledgement from each peer
- Forces turn-taking for "hot" data
  - Users may have to wait for right to edit
- Can't grab locks while offline
  - Must predict what users will want to edit
    - (Or prevent users from editing)

Transactions

- Group modifications into bundles that are committed atomically
- Can avoid lock issues preventing work
  - Use optimistic concurrency control
  - Open a transaction even while offline
  - User can always do work
- Commit transactions via 2-phase protocol
  - Much simpler in the client/server case

Transaction issues

- Can't commit while offline
- Commit is slow in peer-to-peer case
- Must break model into independent pieces
  - Smaller the better, to allow parallel editing
- Transactions may need to be rolled back
  - Difficult user experience
- Only illusion of always being able to edit
  - Works only if users' edits don't overlap

Do/undo/redo

- Invent a global ordering covering all edits
  - Use timestamps, inter-edit dependencies, etc.
- Keep per-copy log of all edits yet applied
  - Log is sorted by the global ordering
- Applying a newly received edit:
  - Undo edits back to correct place in the log
  - Do the new edit
  - Redo edits previously undone, forward
- Approach used by Groove

"Do/undo/redo" issues

- Subject to undo / redo "thrashing"
  - Can be very expensive to apply each new edit
- Difficult to design useful commands
  - Must give good results under heavy reordering
- Can't always preserve intentions
  - Especially for "sequence-like" data models
  - Therefore unusable for document co-editing

Operation Transformation (OT)

- Tag edits with local history before sending
  - What edits had been applied locally when this one was generated?
- Apply incoming edits in order received
  - No undo / redo
  - Application order different at different copies
- Transform edits to account for ignorance
  - What do we know locally that source of edit didn't know?
- Approach used by PlaceWare, Live Meeting

Operation Transformation (OT)

- Tag edits with local history before sending
  - What edits had been applied locally when this one was generated?
- Apply incoming edits in order received
  - No undo / redo
  - Application order different at different copies
- Transform edits to account for ignorance
  - What do we know locally that source of edit didn't know?
- Approach used by PlaceWare, Live Meeting

Transformation example

- You delete a word in the second sentence:
  - *A: delete 7 characters starting at position 65*
- Meanwhile, I delete entire first sentence:
  - *B: delete 42 characters starting at position 1*
- I transform your edit to account for mine:
  - *A': delete 7 characters starting at position 23*
- You do: *A*; *B*
- I do: *B*; *A'*

Fundamental OT building block

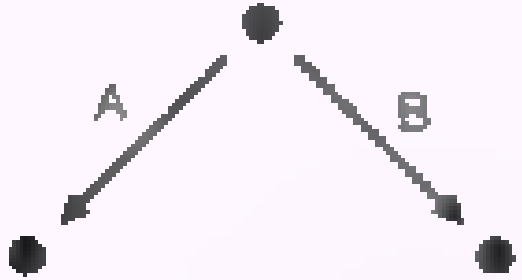
- The transformation diamond:

Fundamental OT building block

- The transformation diamond:

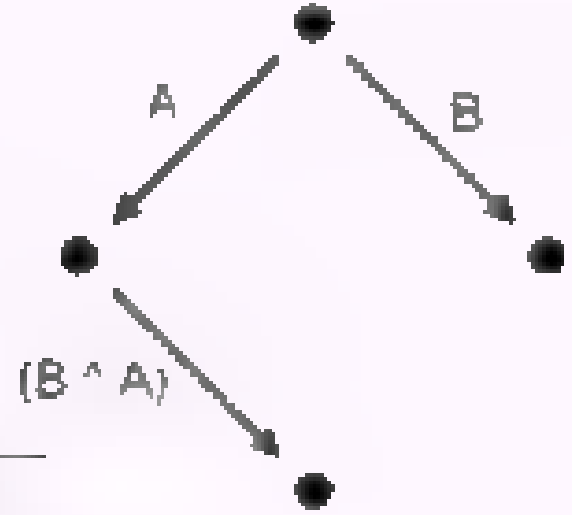
Fundamental OT building block

- The transformation diamond:



Fundamental OT building block

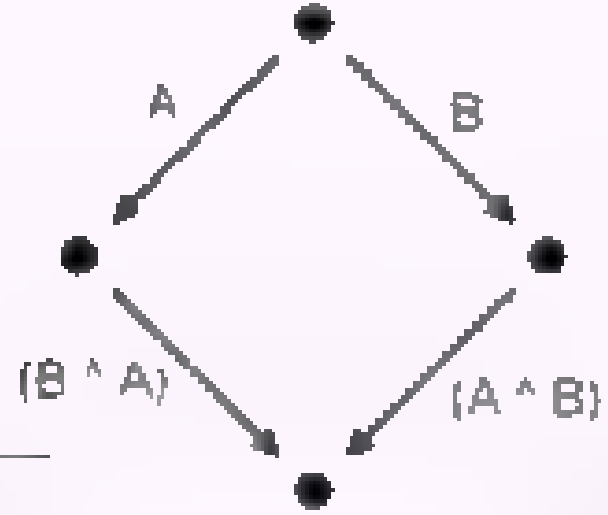
- The transformation diamond:



I transform your B to account for my A and apply it

Fundamental OT building block

- The transformation diamond:

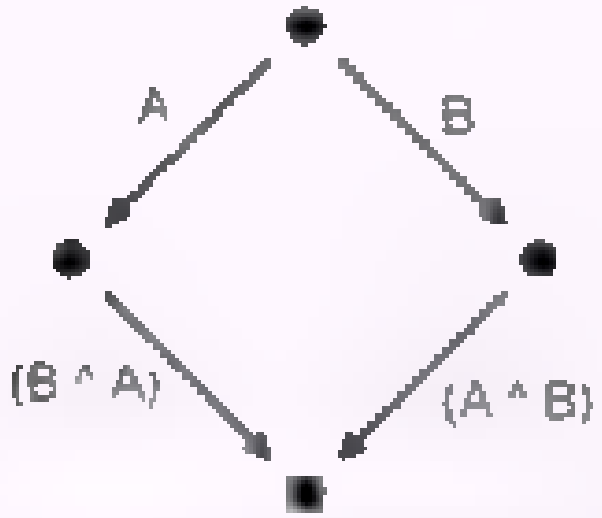


I transform your B to account for my A and apply it

You transform my A to account for your B and apply it

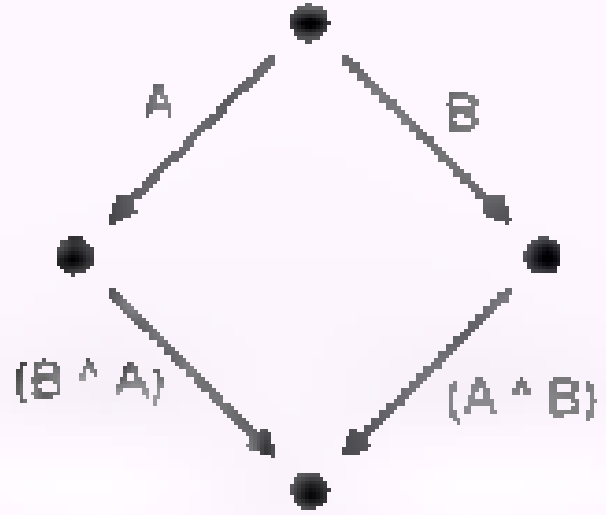
Fundamental OT building block

- The transformation diamond:



Fundamental OT building block

- The transformation diamond:



Convergence Property: A; (B ^ A) = B; (A ^ B)

Knowledge and ignorance

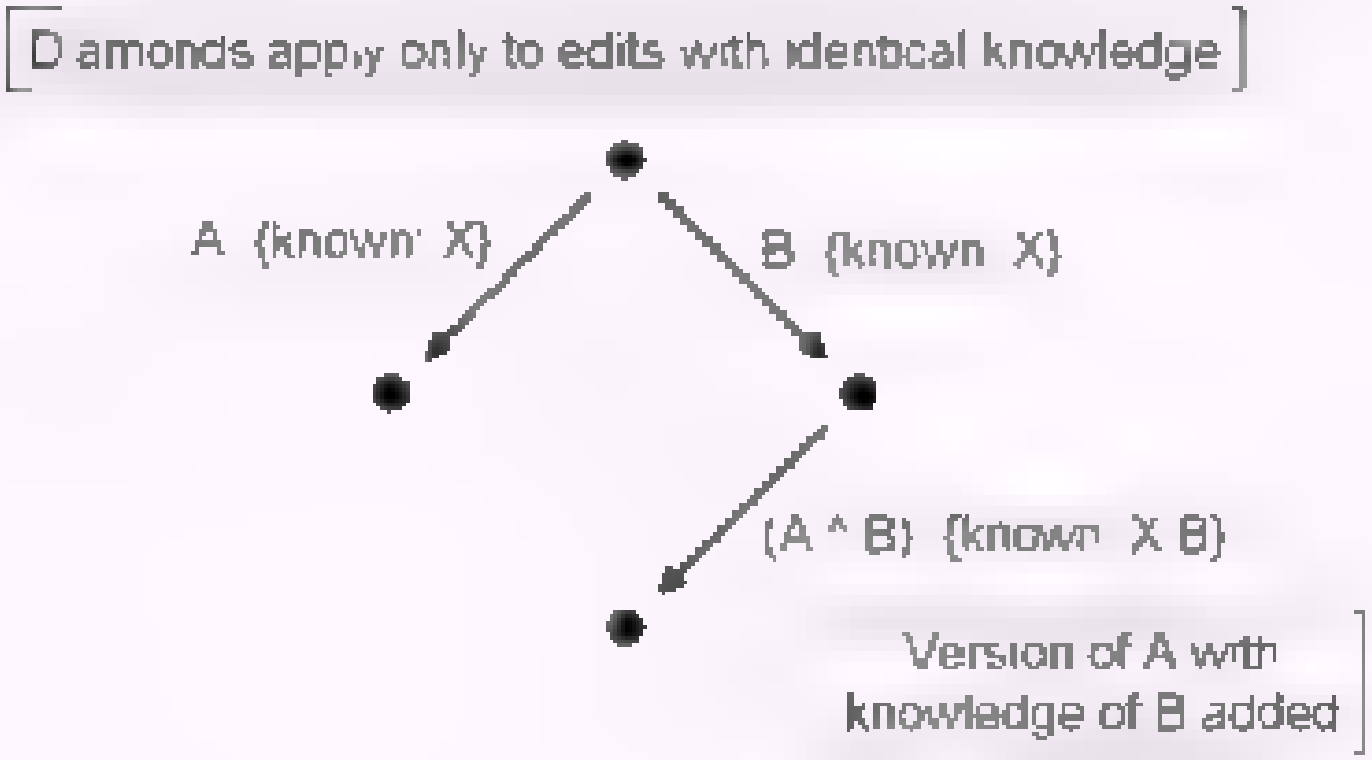
- Did edits A and B originate in same state?
  - Diamonds only work when that's true
- What did that user know when it made B?
  - I.e., what other edits had been applied then?
- Each edit needs a knowledge annotation
- Transformations account for ignorance
  - B was made without knowledge of A; fix it up!

Knowledge in Diamonds

- Diamonds add knowledge to edits:

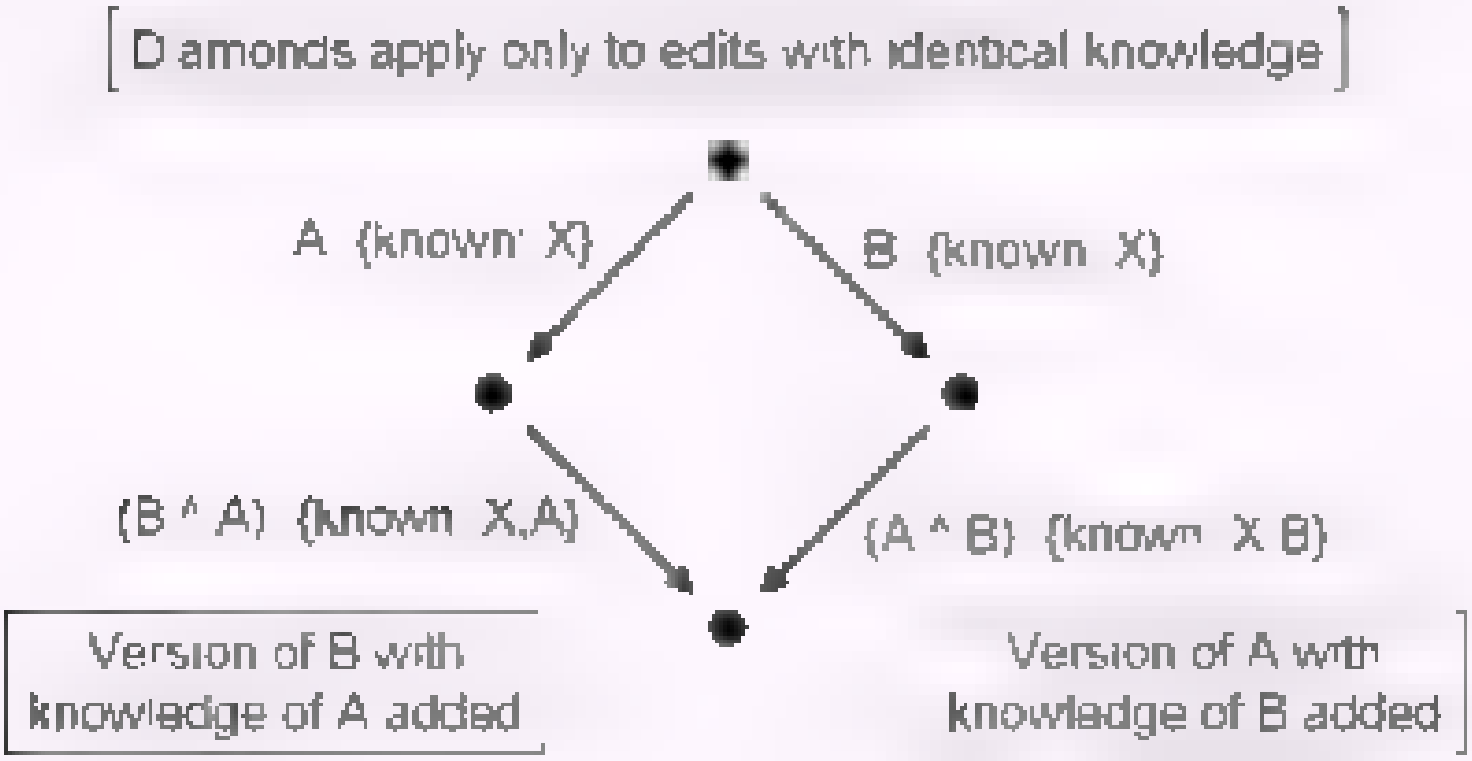
Knowledge in Diamonds

- Diamonds add knowledge to edits:



Knowledge in Diamonds

- Diamonds add knowledge to edits:



Some OT history

- First proposed 1989, by Ellis & Gibbs
  - Bugs in control algorithm (when to transform)
  - Bugs in transformations (how to transform)
- Nichols, Curtis, et al., 1995
  - Correct the bugs, but only for client/server
- Cormack 1995 / Ressel et al. 1996
  - Correct control algorithm for peer-to-peer
    - But incorrect transformations (not even convergence)
- Curtis, Fall 2005
  - Finally, correct transformations
    - Both convergence and intention preservation

OT à la Nichols, Curtis, et al.

- Client/server system
- Communication is ordered and reliable
- Server has separate OT session per client
  - So all sessions have exactly two parties
  - Server copies edits between sessions
    - Receive edit from one client
    - Transform as appropriate
    - Apply it to server's copy of the data
    - To other clients, pretend server originated that transformed edit

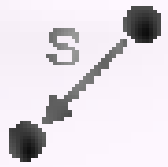
OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:

OT à la Nichols, Curtis, et al.

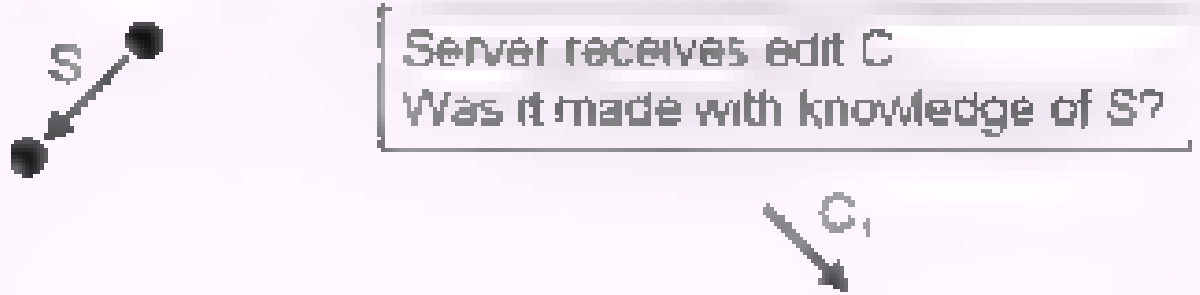
- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage

Server makes edit S



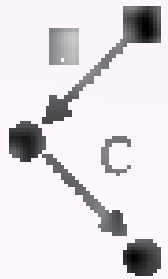
OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:



OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:

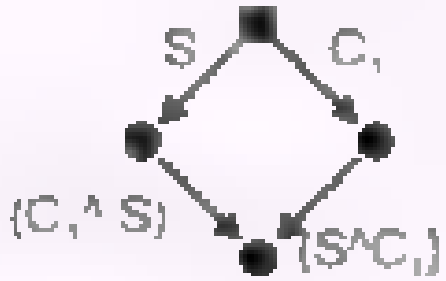


Server receives edit C
Was it made with knowledge of S?

Yes?

OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:



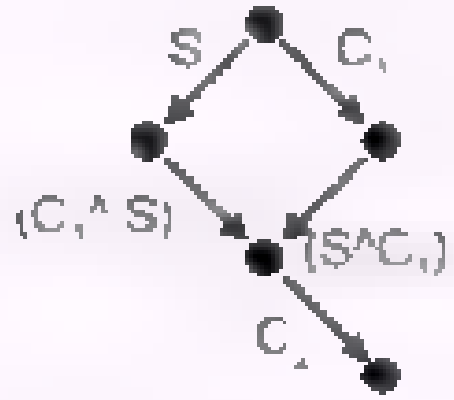
OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:

Server receives edit C_1.
Was it made with knowledge of S?

OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:

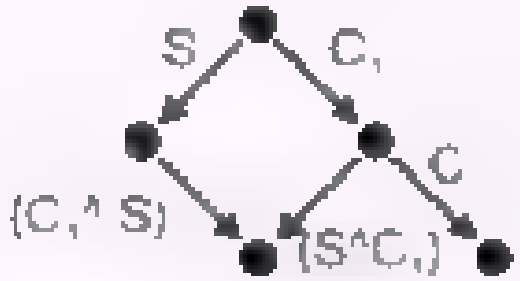


Yes?

Server receives edit C
Was it made with knowledge of S?

OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:

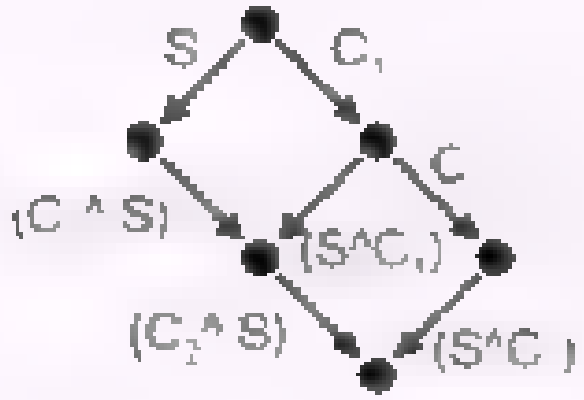


No?

Server receives edit C
Was it made with knowledge of S?

OT à la Nichols, Curtis, et al.

- Knowledge is easy to represent
  - How many of my edits had you seen when you did that?
- Diamonds are easy to manage:



OT à la everybody else (including Moby)

- Peer-to-peer system
- Communication is unordered, unreliable
- Session contains K parties ($K \geq 1$)
  - Every copy knows about ($K - 1$) other copies
- Knowledge is trickier to represent
- Diamonds are much trickier to manage
  - How are you at K-dimensional visualization?

Components of a solution

- Peer-to-peer communications
  - Messages containing edits, roster changes
- Adding reliability and causality
  - Representing knowledge
- Accounting for ignorance
  - Finding useful diamonds to compute
- Defining correct transformations

Communications

- Most messages should be "broadcast" to all users
  - Not necessarily delivered quickly, in order, or at all!
- Lots of options available
  - Full-mesh TCP or UDP
  - IP multicast UDP
  - Groove-like relay server in the cloud
  - Email
  - Shared file (!)

Respecting causality

- Messages can arrive before ones they depend on
  - Edit A is made with knowledge of B, but A arrives before B does
- Tag messages with *version vector*
  - For each user, how many of that user's edits had you seen when you made this?
- Receiver queues up early messages
  - Waiting to receive everything it knew about

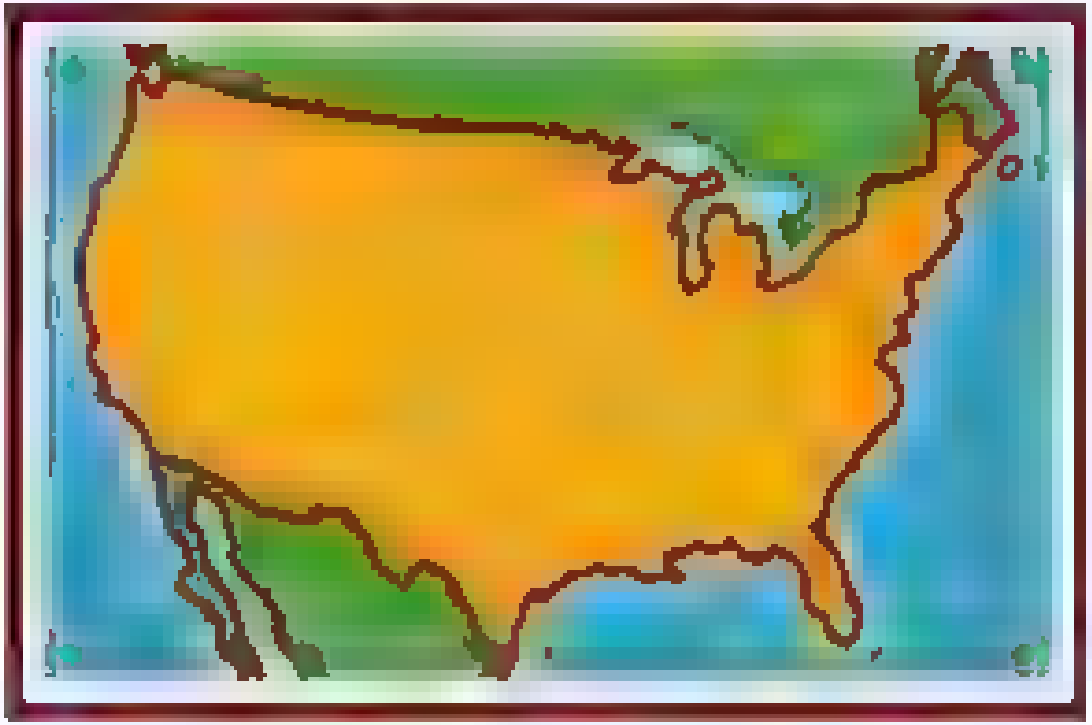
Achieving reliable delivery

- Messages can be lost in transit…
- …but every user must get every message
  - Eventually…
- Periodically broadcast your knowledge
  - See if others know messages you don't
  - Request message from someone (anyone!)
- Alternative: send acks for messages
  - Wait a while, to bundle acks together
  - Can result in fewer messages overall

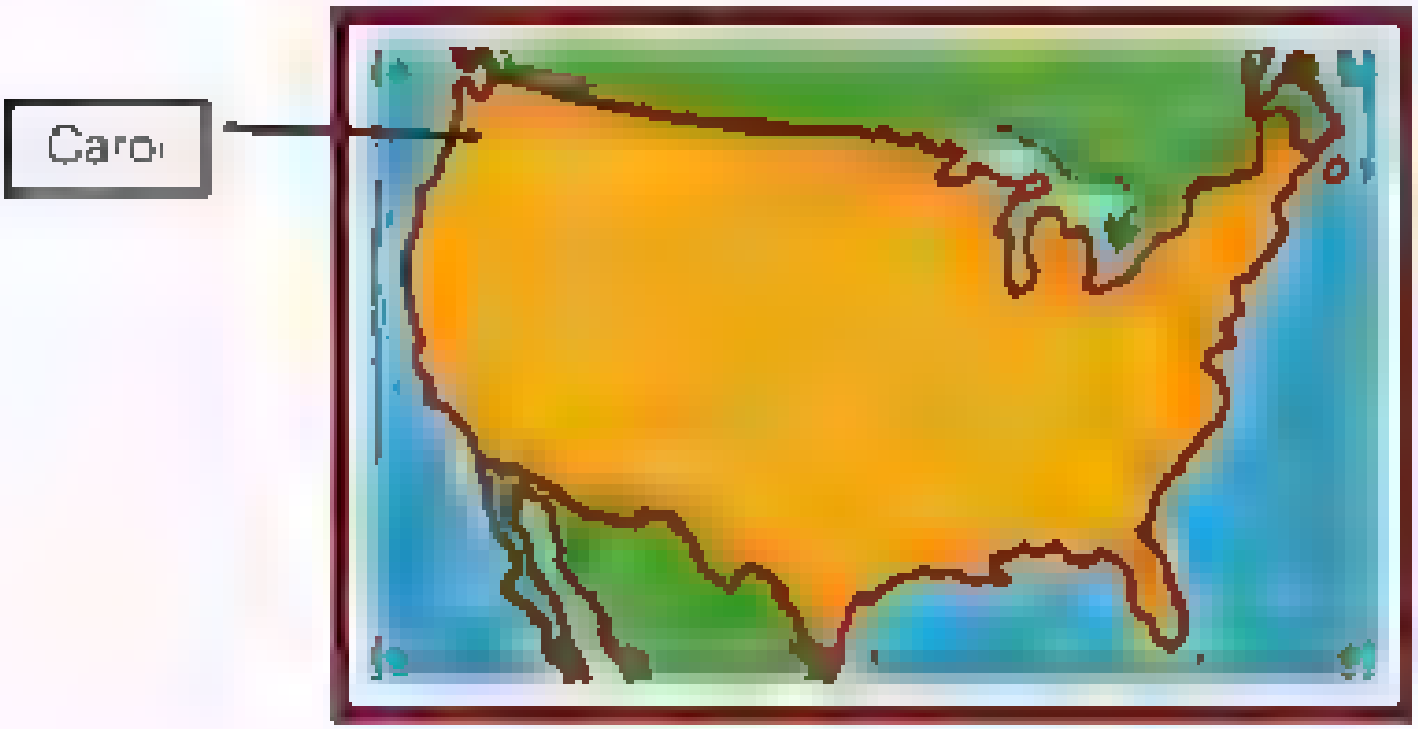
Accounting for ignorance

- *K*-party case is harder than 2-party case
  - A's knowledge and B's can be incomparable
  - Diamonds demand A and B have identical knowledge
- Let's look at an example…

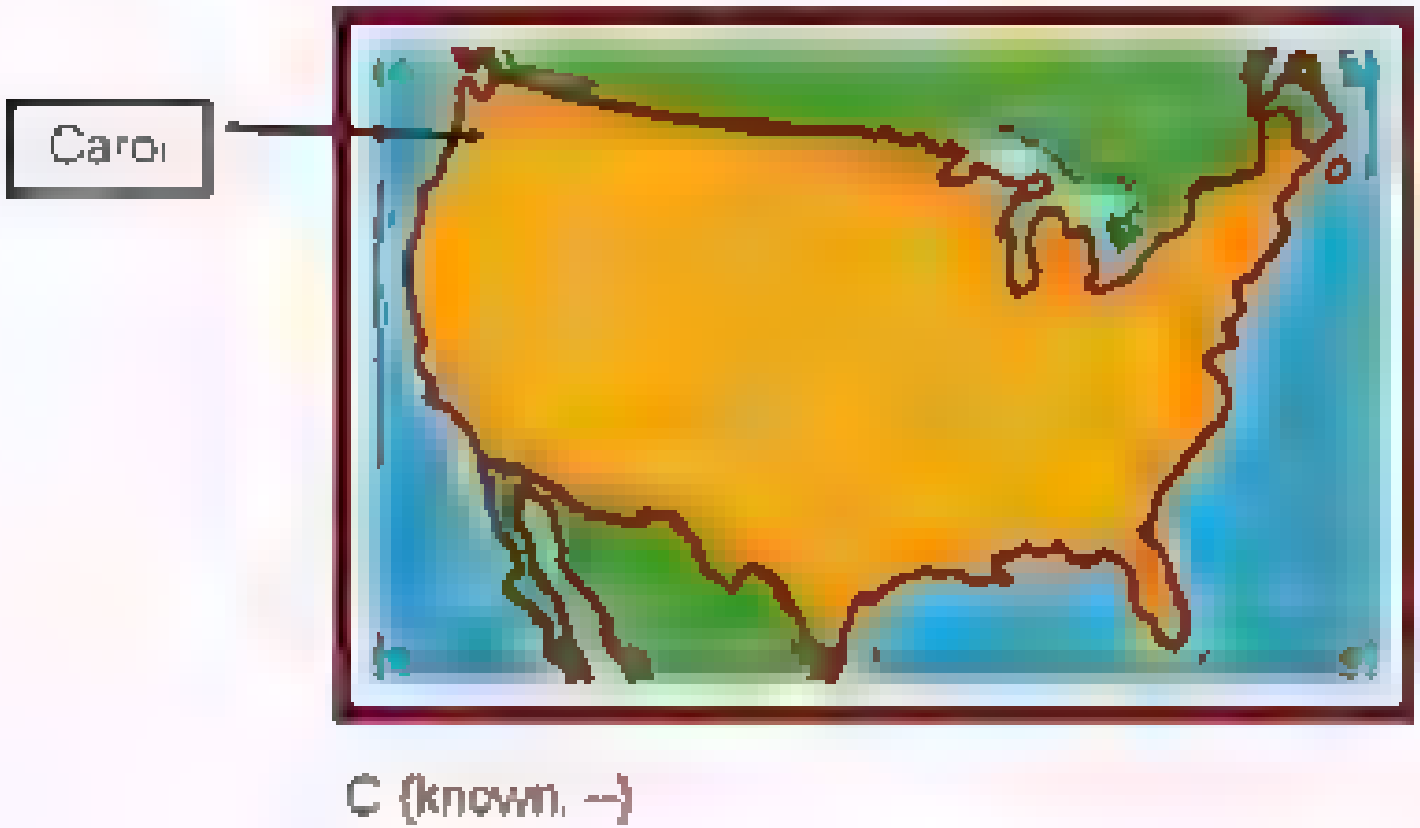
A modest example



A modest example

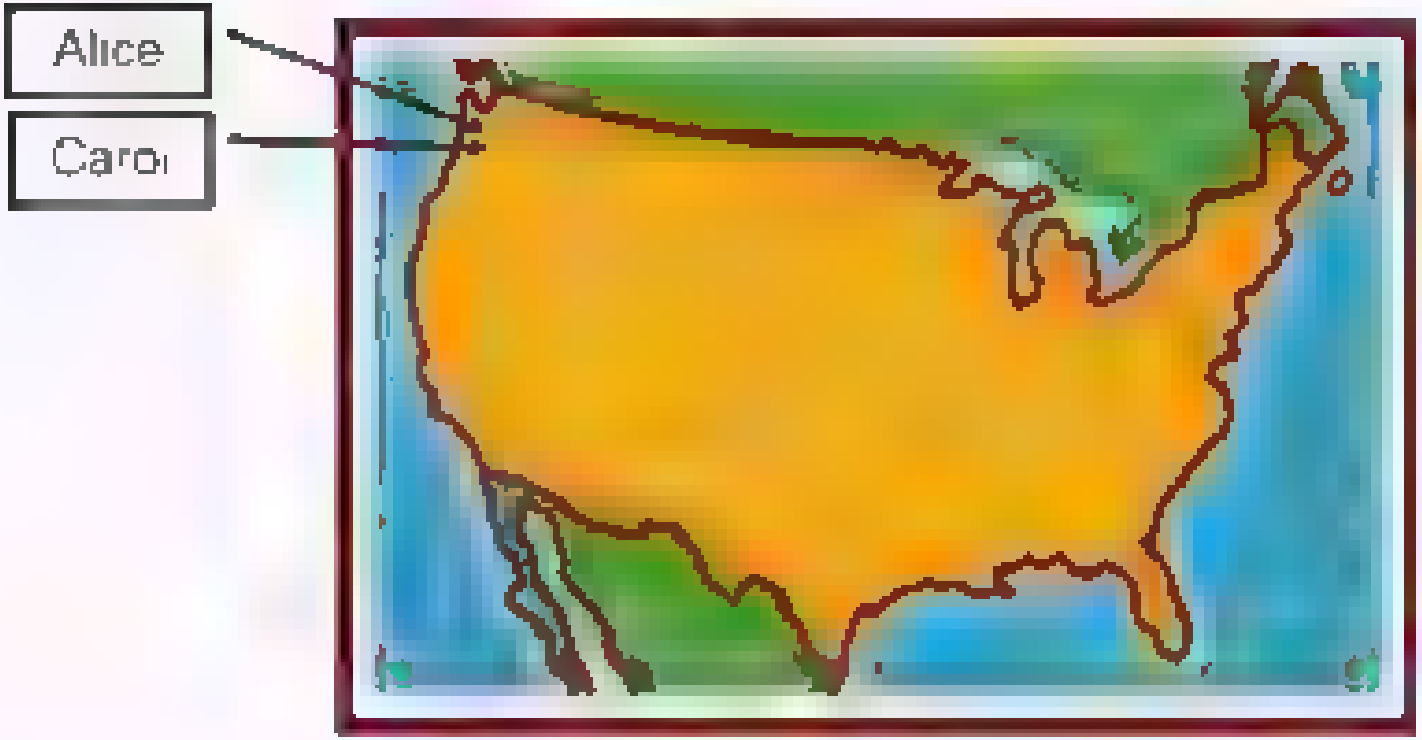


A modest example



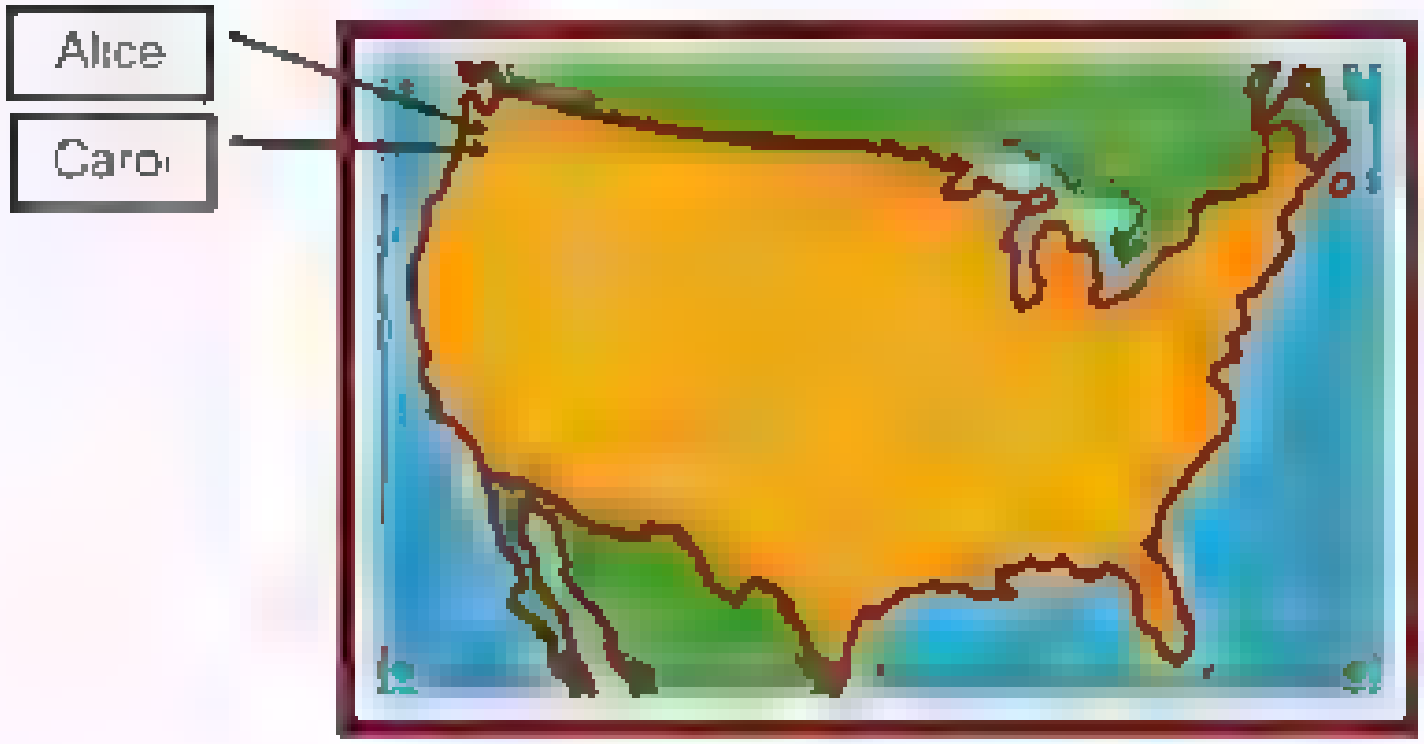
C (known, --)

A modest example



C {known, --}

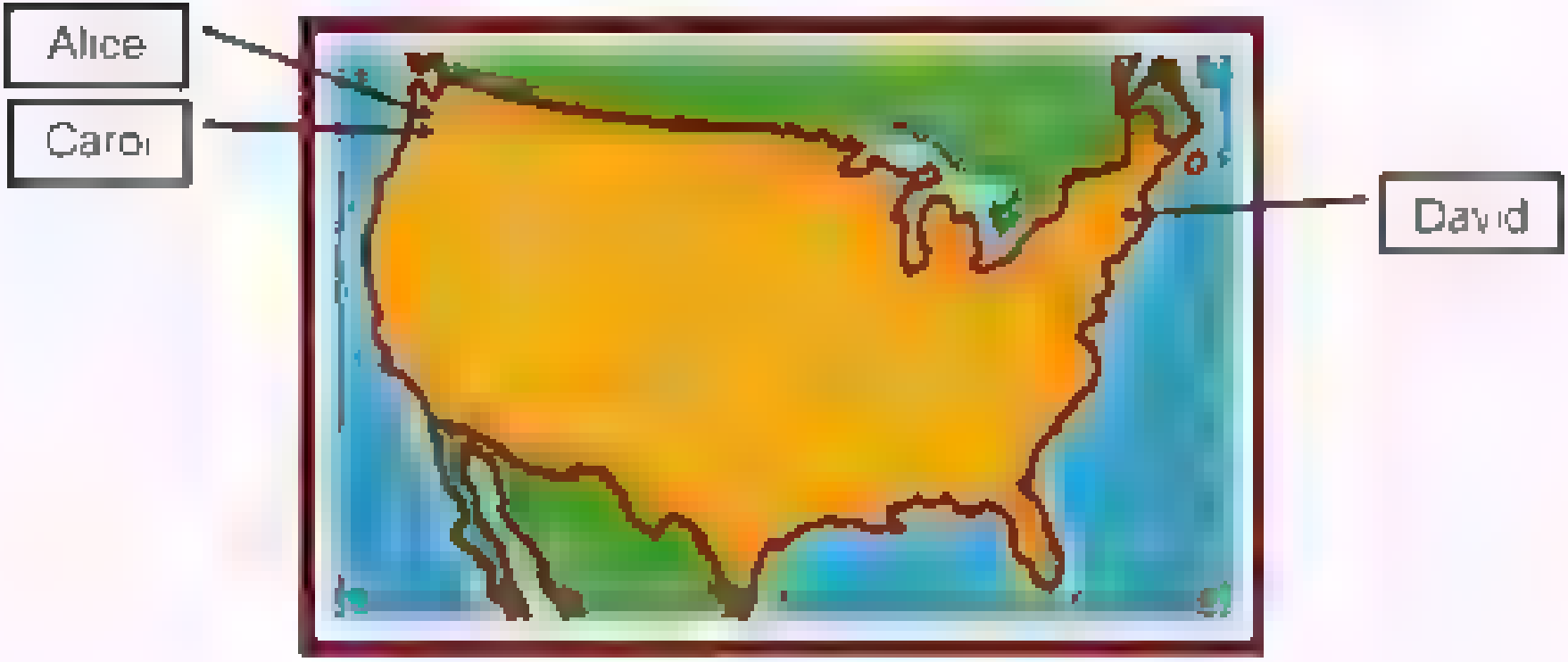
A modest example



C {known: · }

A {known: C}

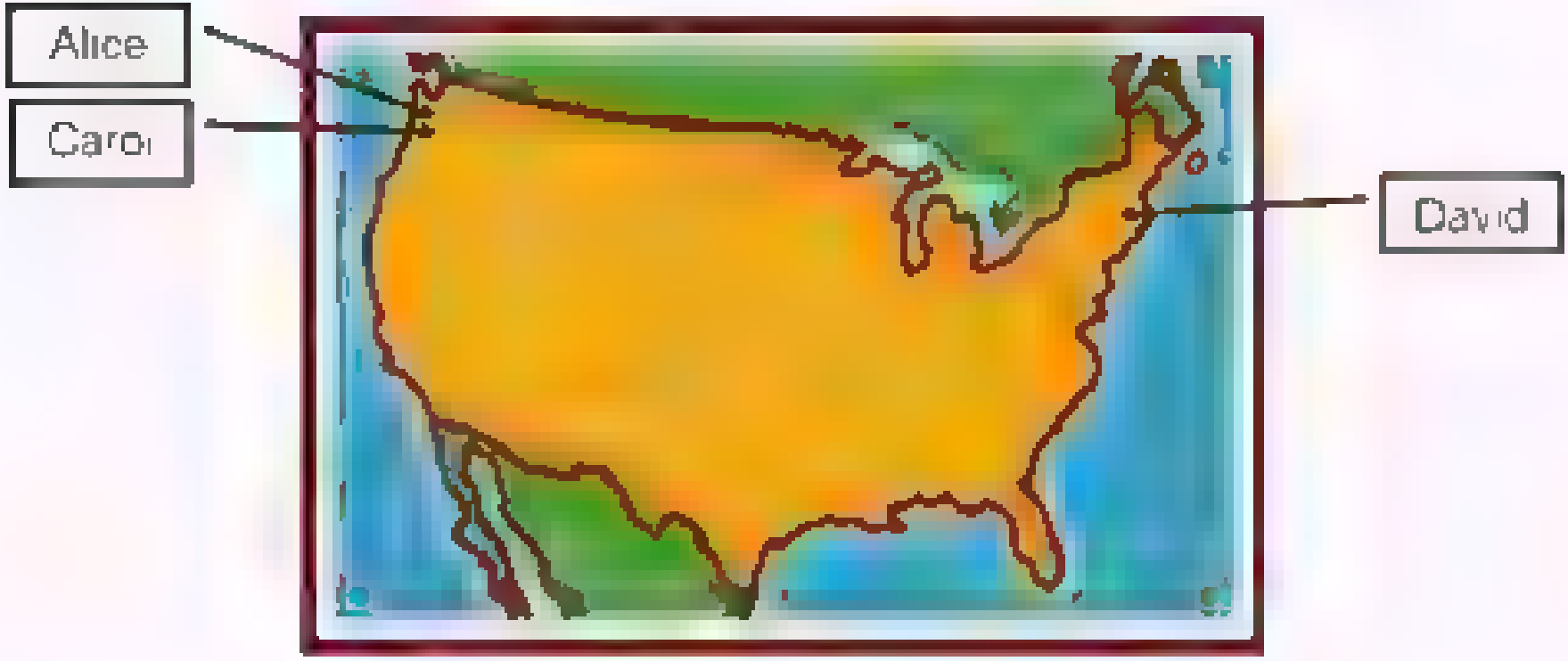
A modest example



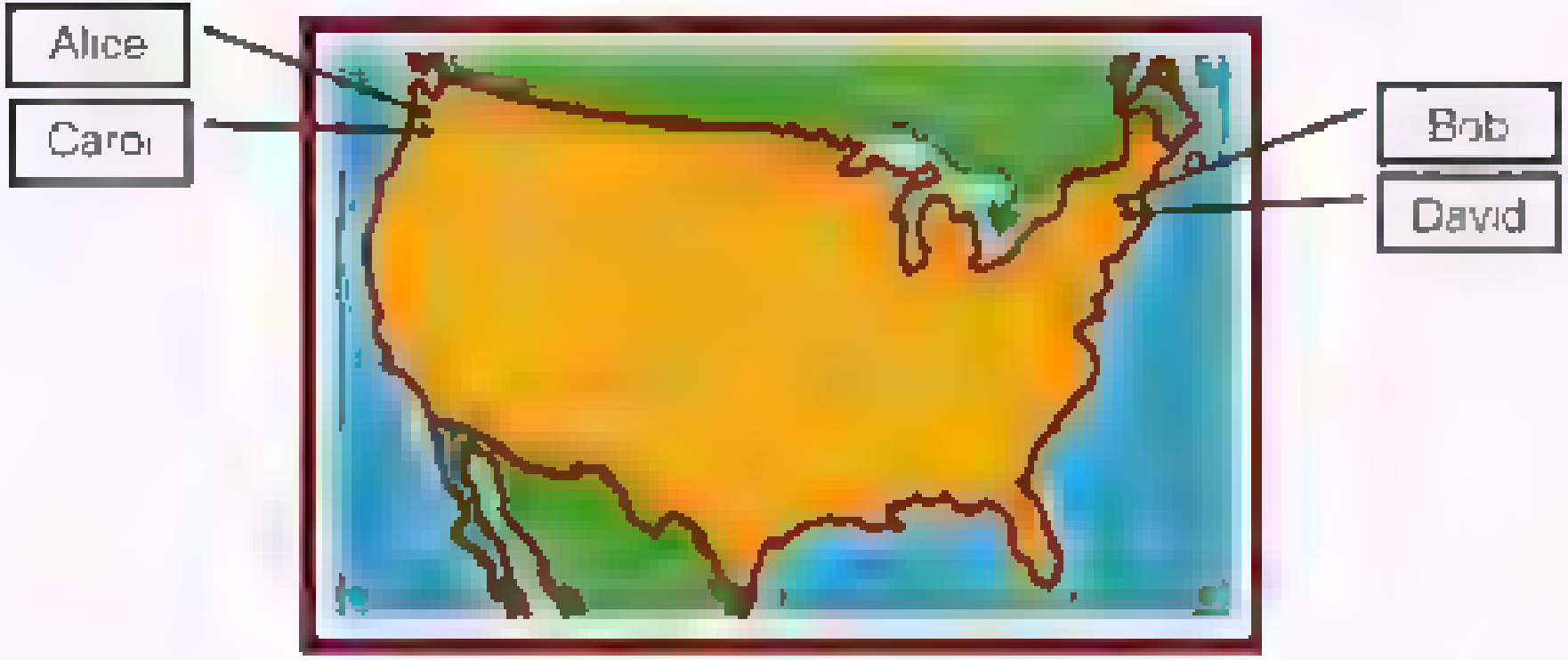
C {known: --}

A {known: C}

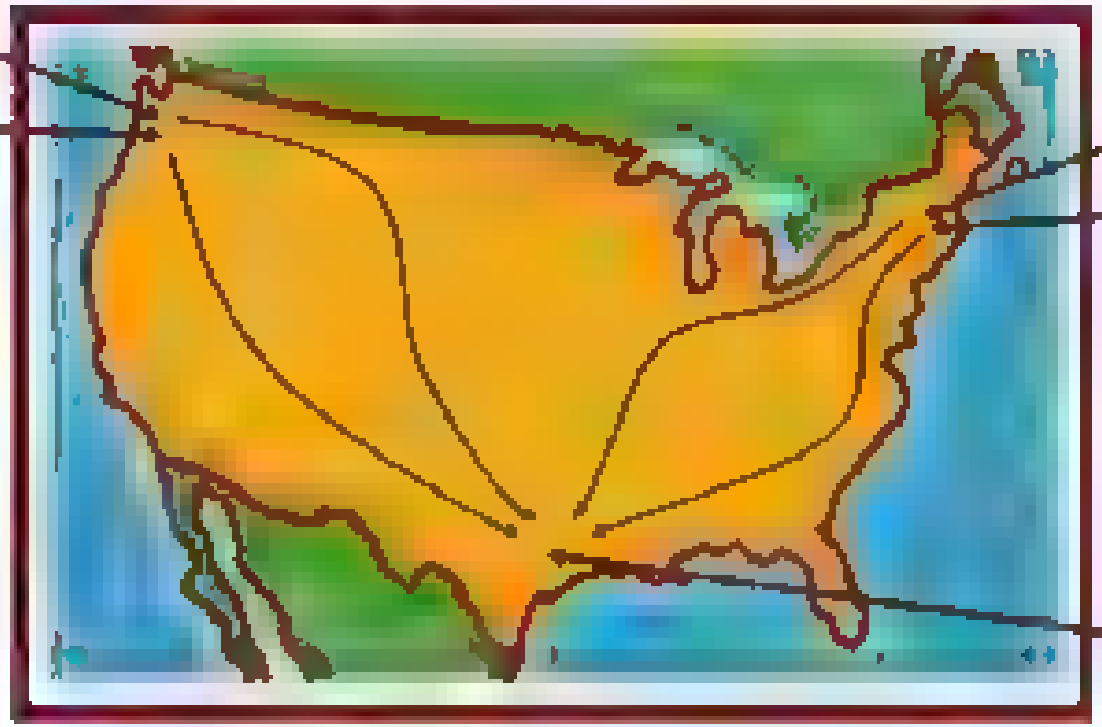
A modest example



A modest example



A modest example



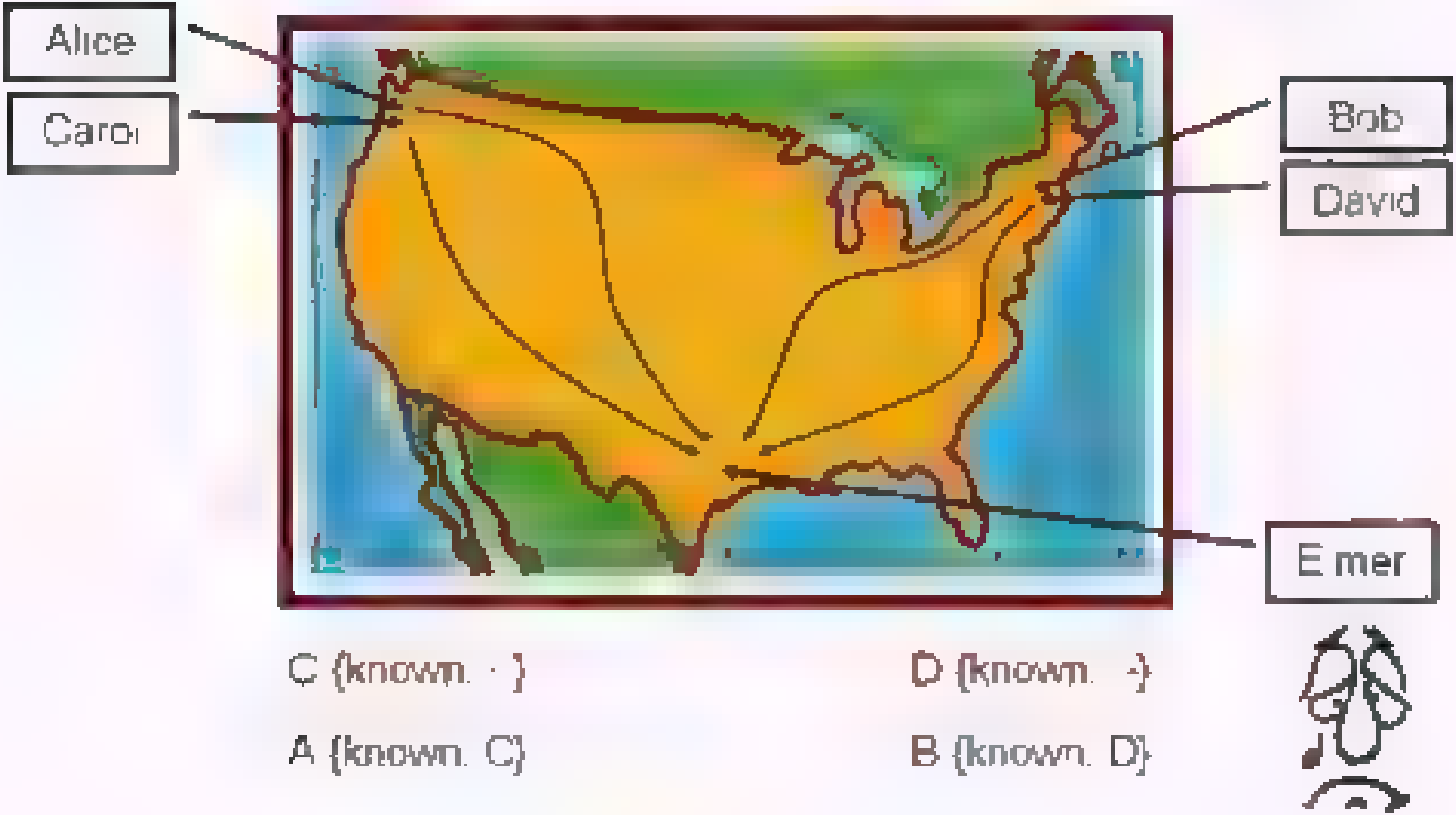
C {known: · }

A {known: C}

D {known: }

B {known: D}

A modest example



What would Elmer do?

What would Elmer do?

Received edits

What would Elmer do?

Received edits

Elmer's knowledge

--

What would Elmer do?

Received edits

C {known --}

Elmer's knowledge

--

What would Elmer do?

Received edits

C {known --}

C {known --}

Elmer's knowledge

--

What would Elmer do?

Received edits

C {known -->}

Elmer's knowledge

C

What would Elmer do?

Received edits

C {known =}

B {known D}

Elmer's knowledge

C

What would Elmer do?

Received edits

C {known= --}

D {known --}

Elmer's knowledge

C

What would Elmer do?

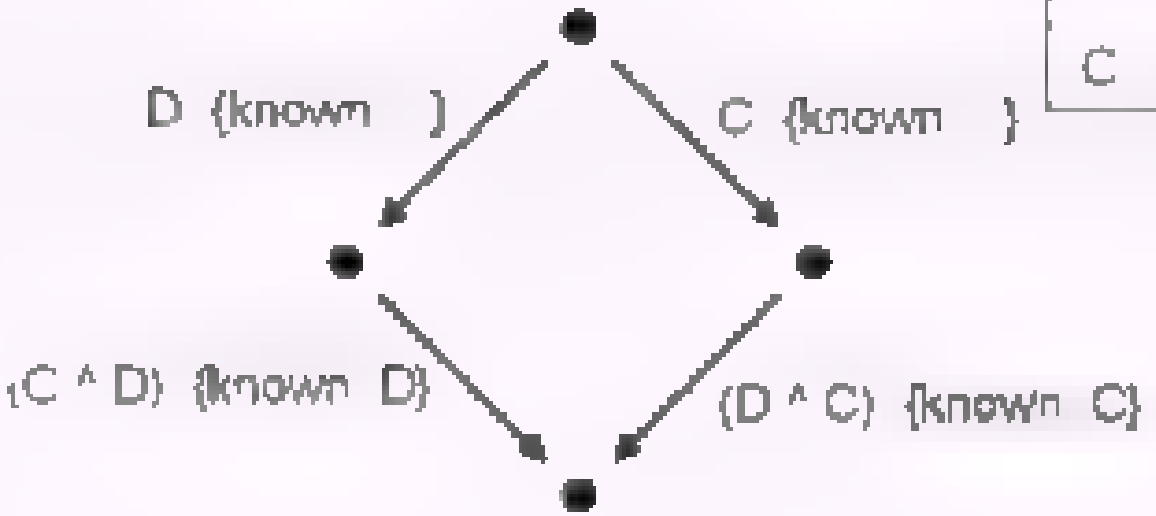
Received edits

C {known: --}

D {known: }

Elmer's knowledge

C



What would Elmer do?

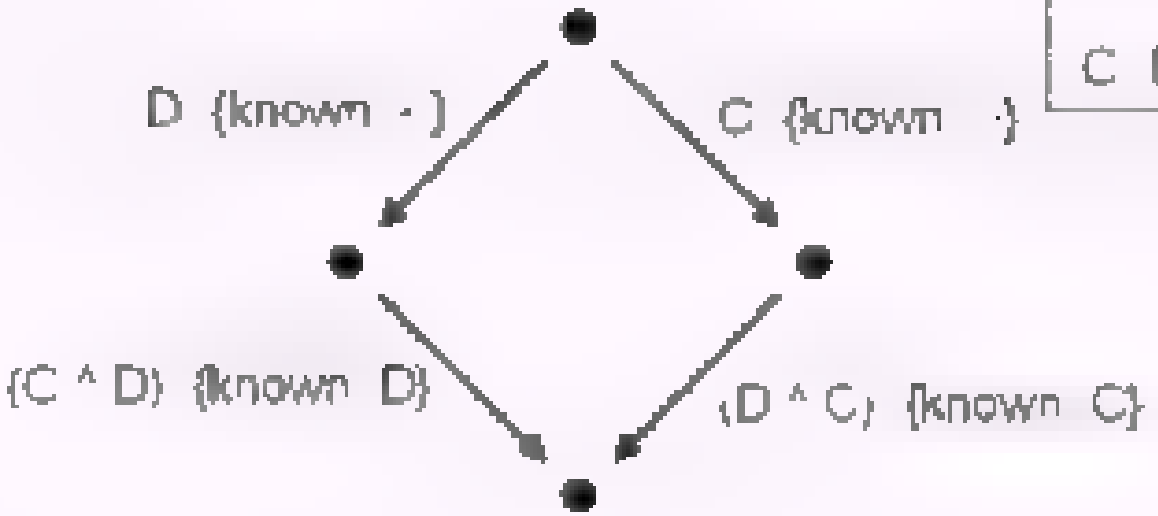
Received edits

C {known: --}

D {known: }

Elmer's knowledge

C D



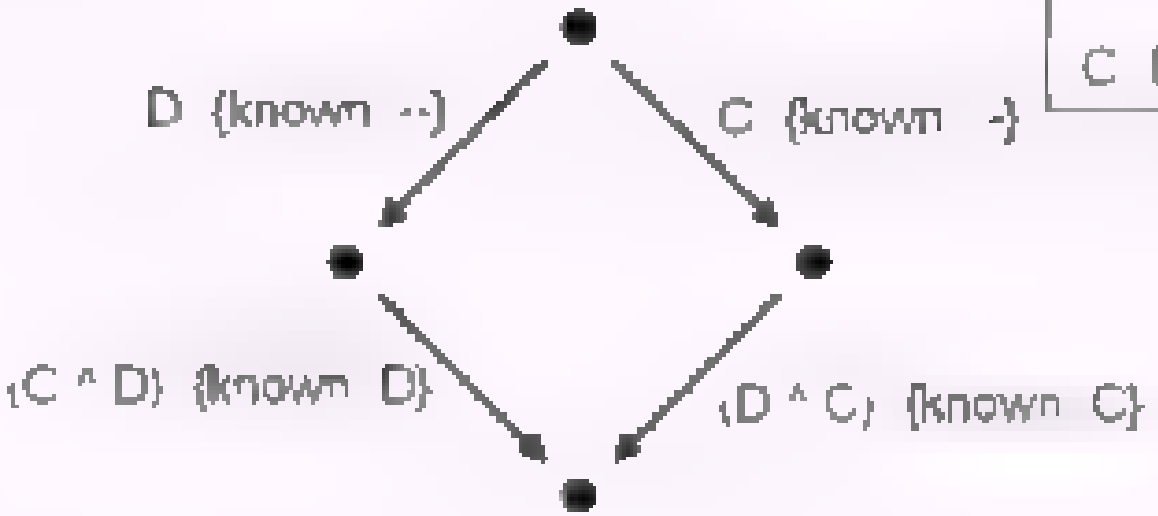
What would Elmer do?

Received edits

C {known* --}

B {known D}

D {known* - }



Elmer's knowledge

C D

What would Elmer do?

Received edits

C {known* --}

B {known D}

D {known* }

Elmer's knowledge

C D

What would Elmer do?

Received edits

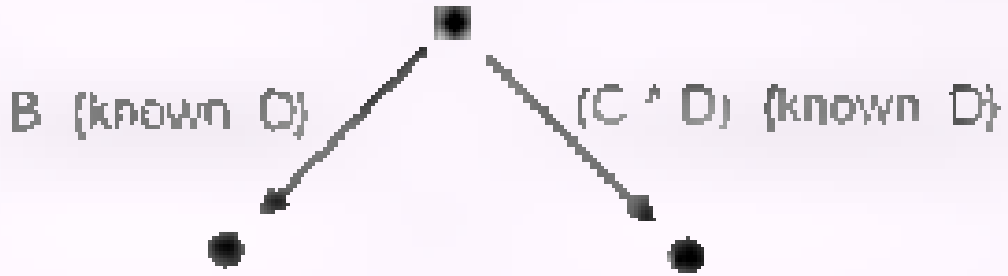
C {known --}

B {known D}

D {known --}

Elmer's knowledge

C D



What would Elmer do?

Received edits

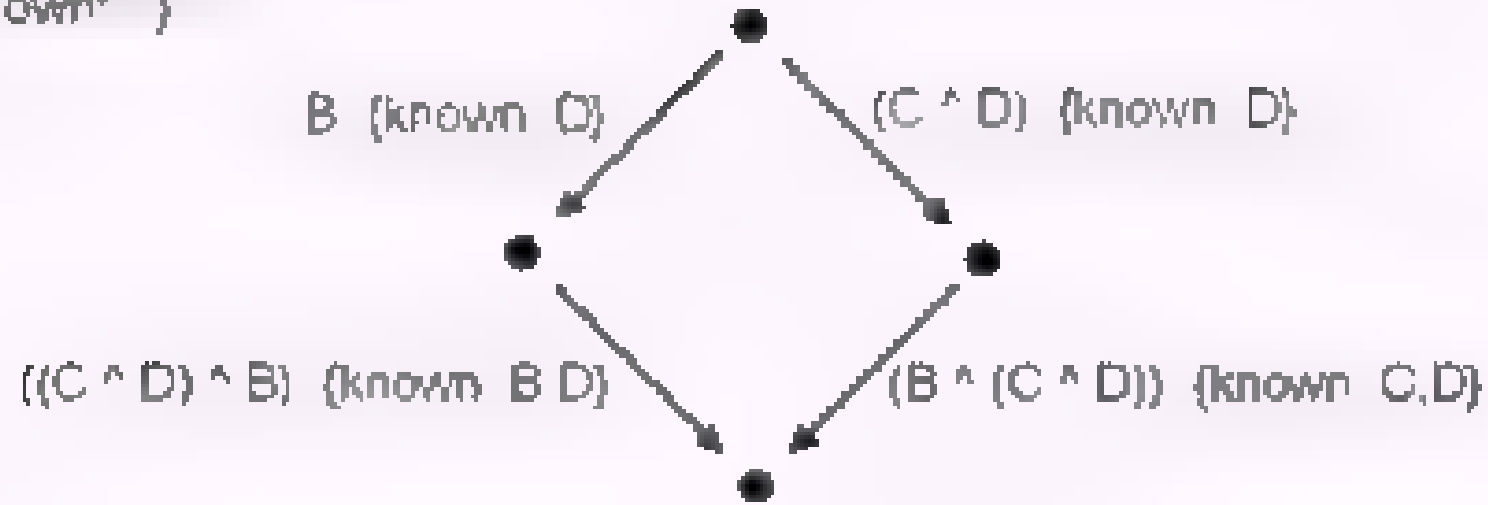
C {known: --}

B {known: D}

D {known: }

Elmer's knowledge

C D



What would Elmer do?

Received edits

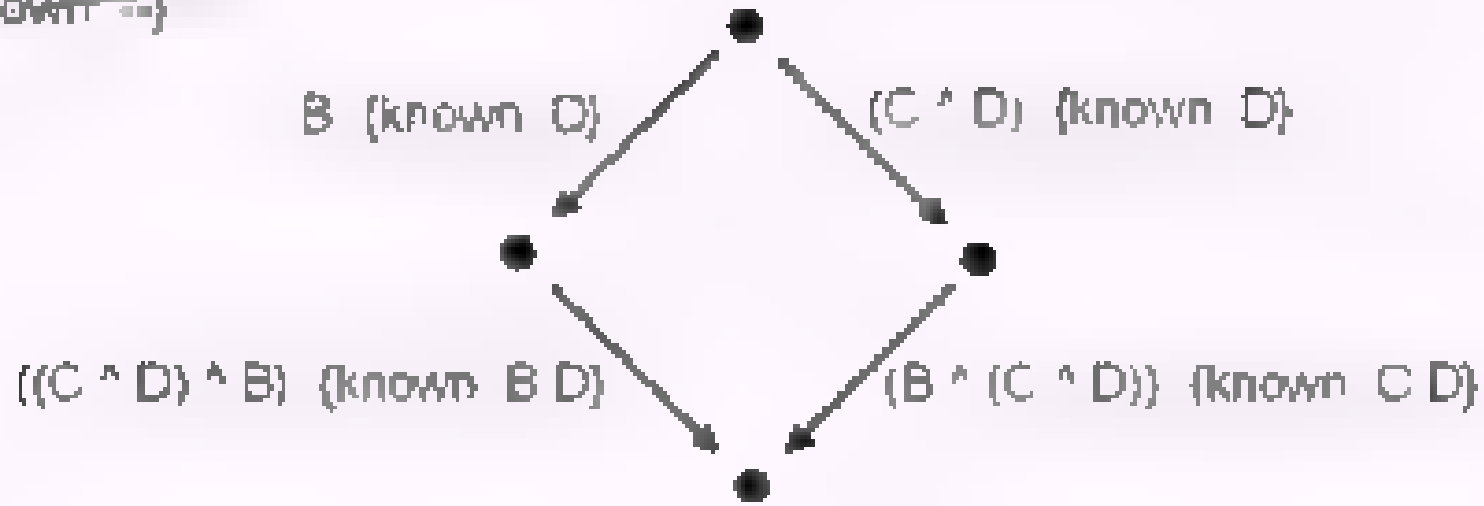
C {known --}

B {known D}

D {known --}

Elmer's knowledge

C D B



What would Elmer do?

Received edits

C {known =}

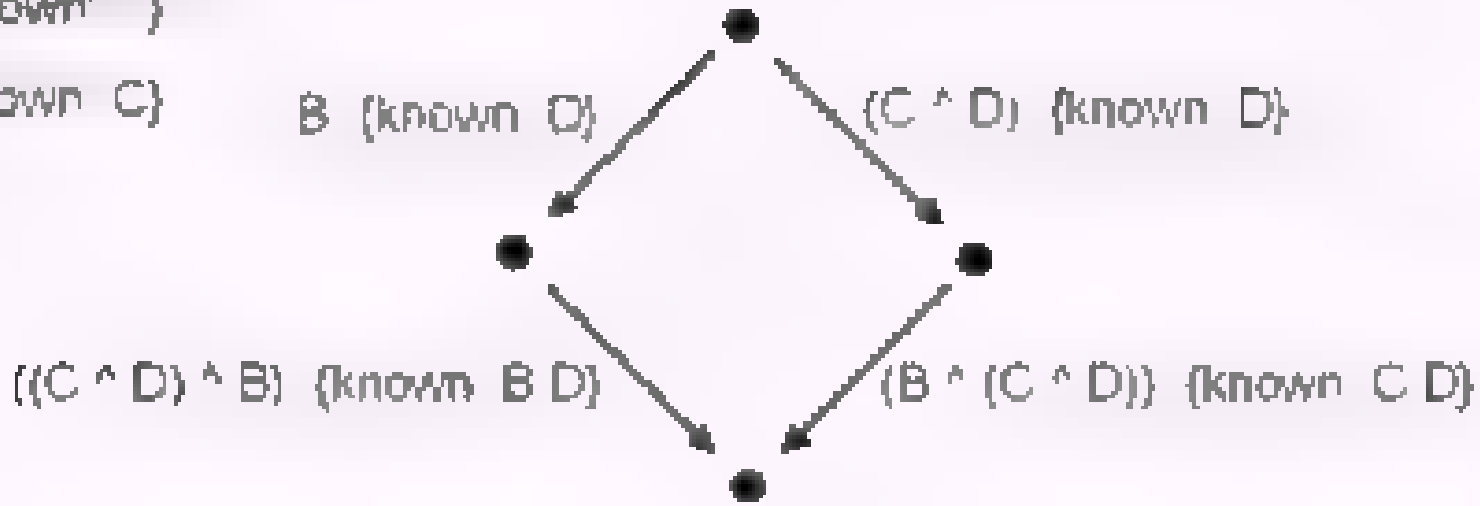
B {known D}

D {known }

A {known C}

Elmer's knowledge

C D B



What would Elmer do?

Received edits

C {known: --}

B {known: D}

D {known: }

A {known: C}

Elmer's knowledge

C D B

What would Elmer do?

Received edits

C {known: --}

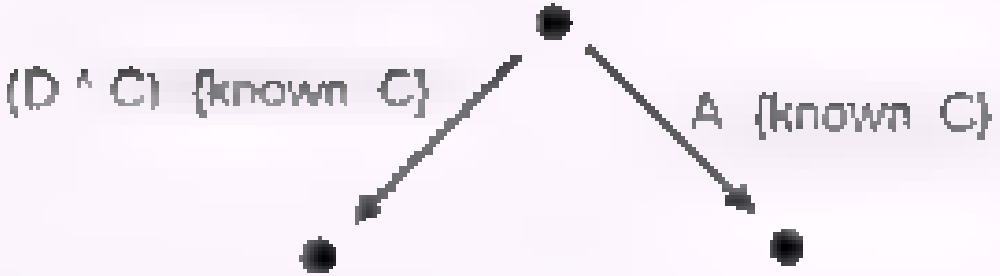
B {known: D}

D {known: }

A {known: C}

Elmer's knowledge

C D B



What would Elmer do?

Received edits

C {known: --}

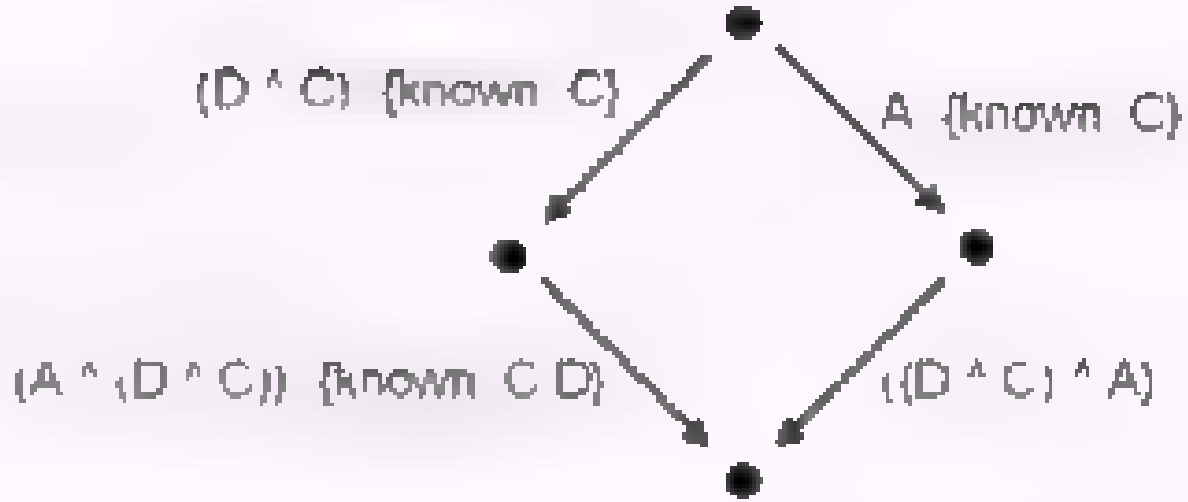
B {known: D}

D {known: --}

A {known: C}

Elmer's knowledge

C D B



What would Elmer do?

Received edits

C {known: --}

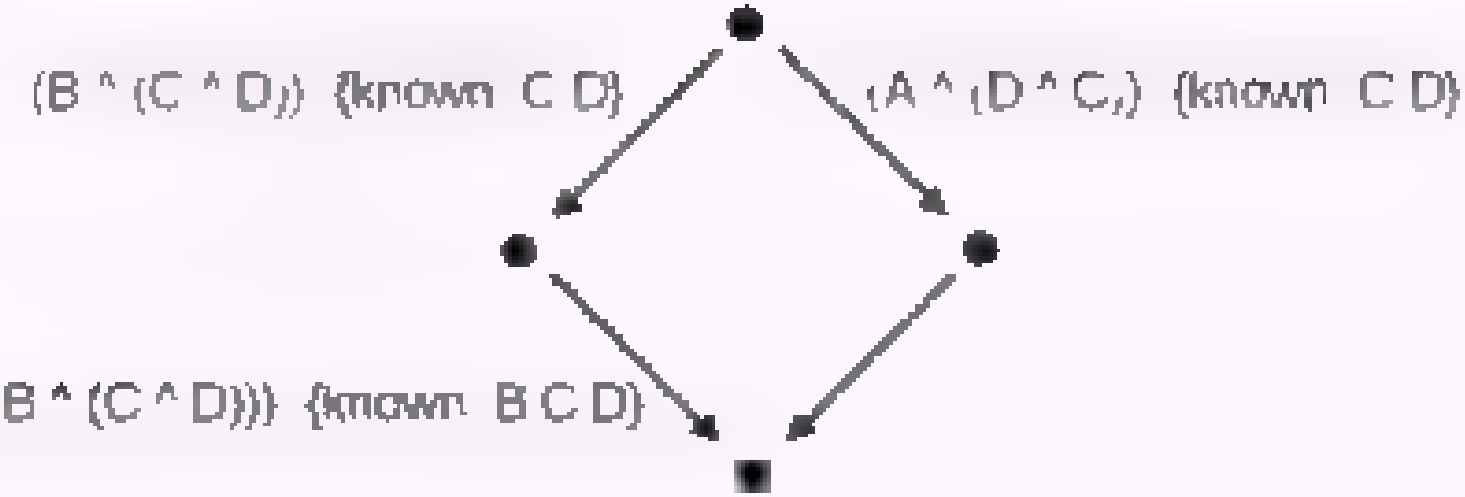
B {known: D}

D {known: }

A {known: C}

Elmer's knowledge

C D B



What would Elmer do?

Received edits

C {known: --}

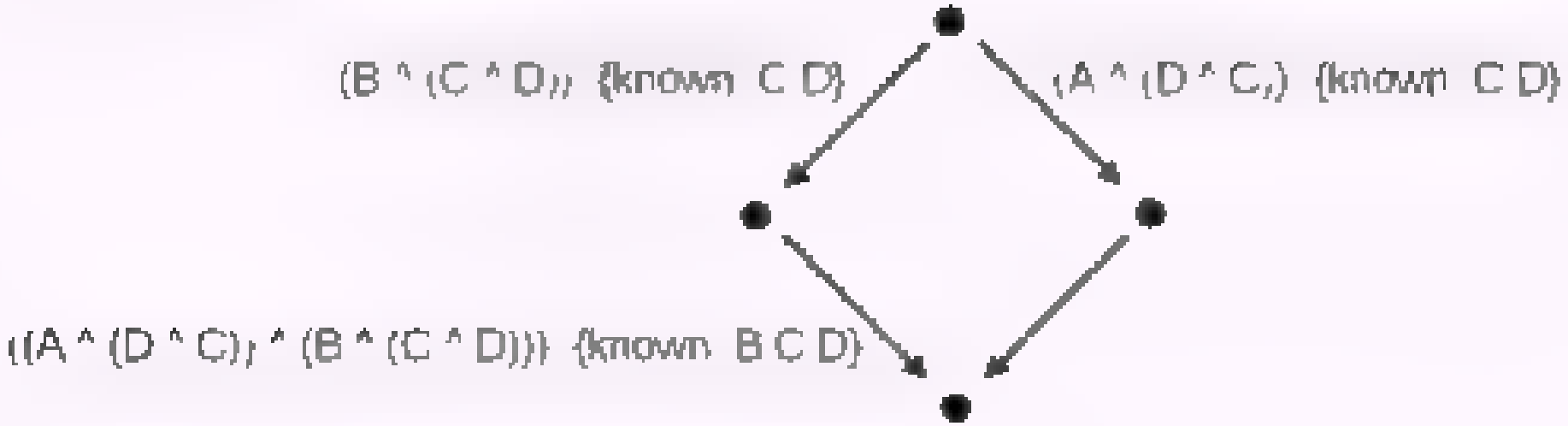
B {known: D}

D {known: }

A {known: C}

Elmer's knowledge

C D B A



What would Elmer do?

Received edits

C {known=--}

B {known=D}

D {known=}

A {known=C}

Elmer's knowledge

C D B A

What would Elmer do?

Received edits

C {known=}

B {known=D}

D {known=}

A {known=C}

Elmer's knowledge

C D B A

Whew!

What would Elmer do?

Received edits

C {known: --}

B {known: D}

D {known: }

A {known: C}

Elmer's knowledge

C D B A

Whew!

But can we <u>always</u> find such useful diamonds?

Solving one step of ignorance

Solving one step of ignorance

- Suppose we receive new edit A
  - Assume A is ignorant of some other edits we know

Solving one step of ignorance

- Suppose we receive new edit A
  - Assume A is ignorant of some other edits we know
- Find an edit B that A doesn't know, where B is ignorant of everything else that A doesn't know
  - Must be at least one such no knowledge cycles

Solving one step of ignorance

- Suppose we receive new edit A
  - Assume A is ignorant of some other edits we know
- Find an edit B that A doesn't know, where B is ignorant of everything else that A doesn't know
  - Must be at least one such no knowledge cycles
- Recursively compute B′
  - Version of B that knows exactly what A knows

Solving one step of ignorance

- Suppose we receive new edit A
  - Assume A is ignorant of some other edits we know
- Find an edit B that A doesn't know, where B is ignorant of everything else that A doesn't know
  - Must be at least one such no knowledge cycles
- Recursively compute B′
  - Version of B that knows exactly what A knows
- Compute A′ = (A ^ B′)
  - A′ is one edit less ignorant than A is

Solving one step of ignorance

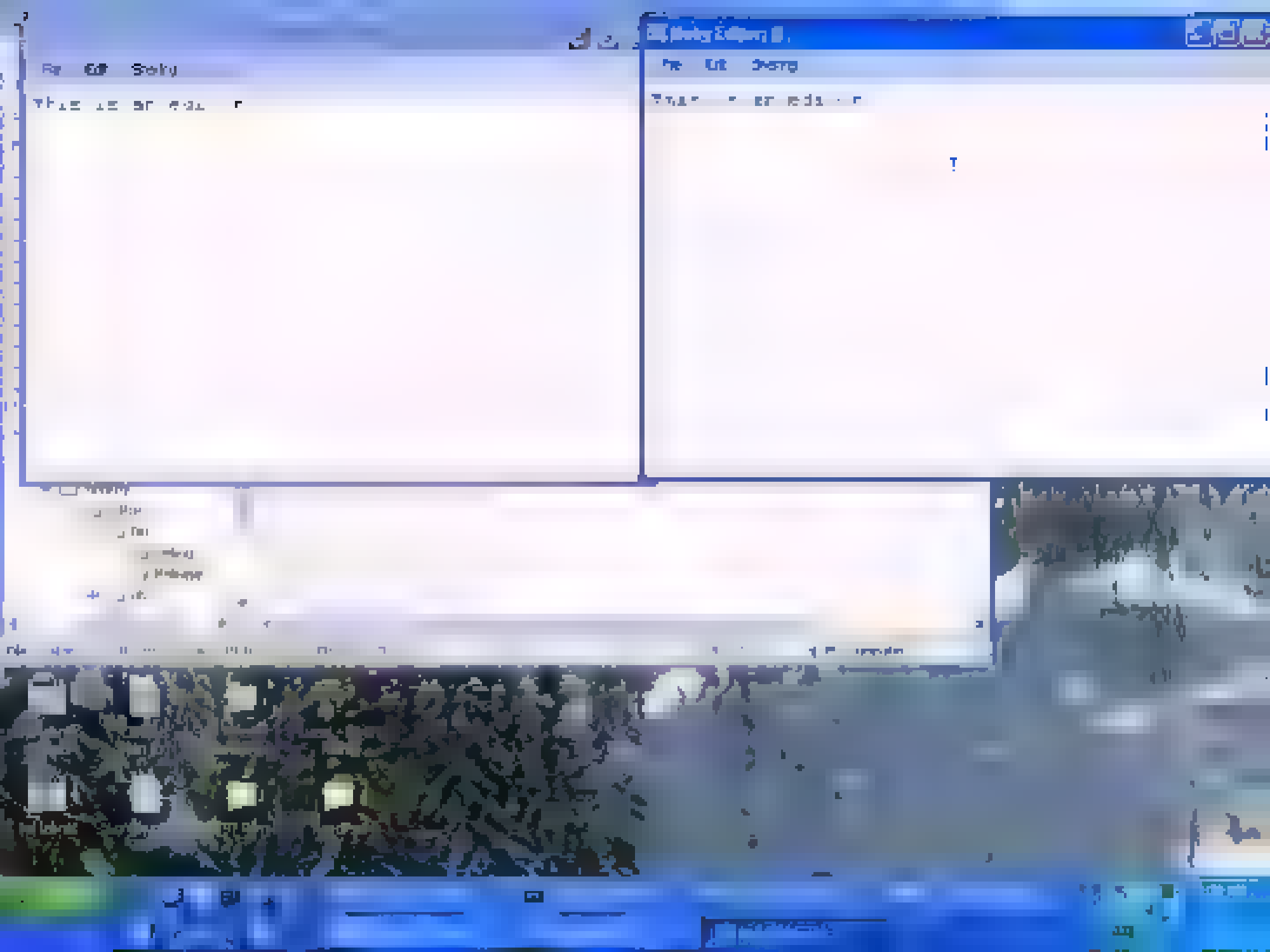
- Suppose we receive new edit A
  - Assume A is ignorant of some other edits we know
- Find an edit B that A doesn't know, where B is ignorant of everything else that A doesn't know
  - Must be at least one such no knowledge cycles
- Recursively compute B′
  - Version of B that knows exactly what A knows
- Compute A′ = (A ^ B′)
  - A′ is one edit less ignorant than A is
- Keep going until no ignorance left, then apply

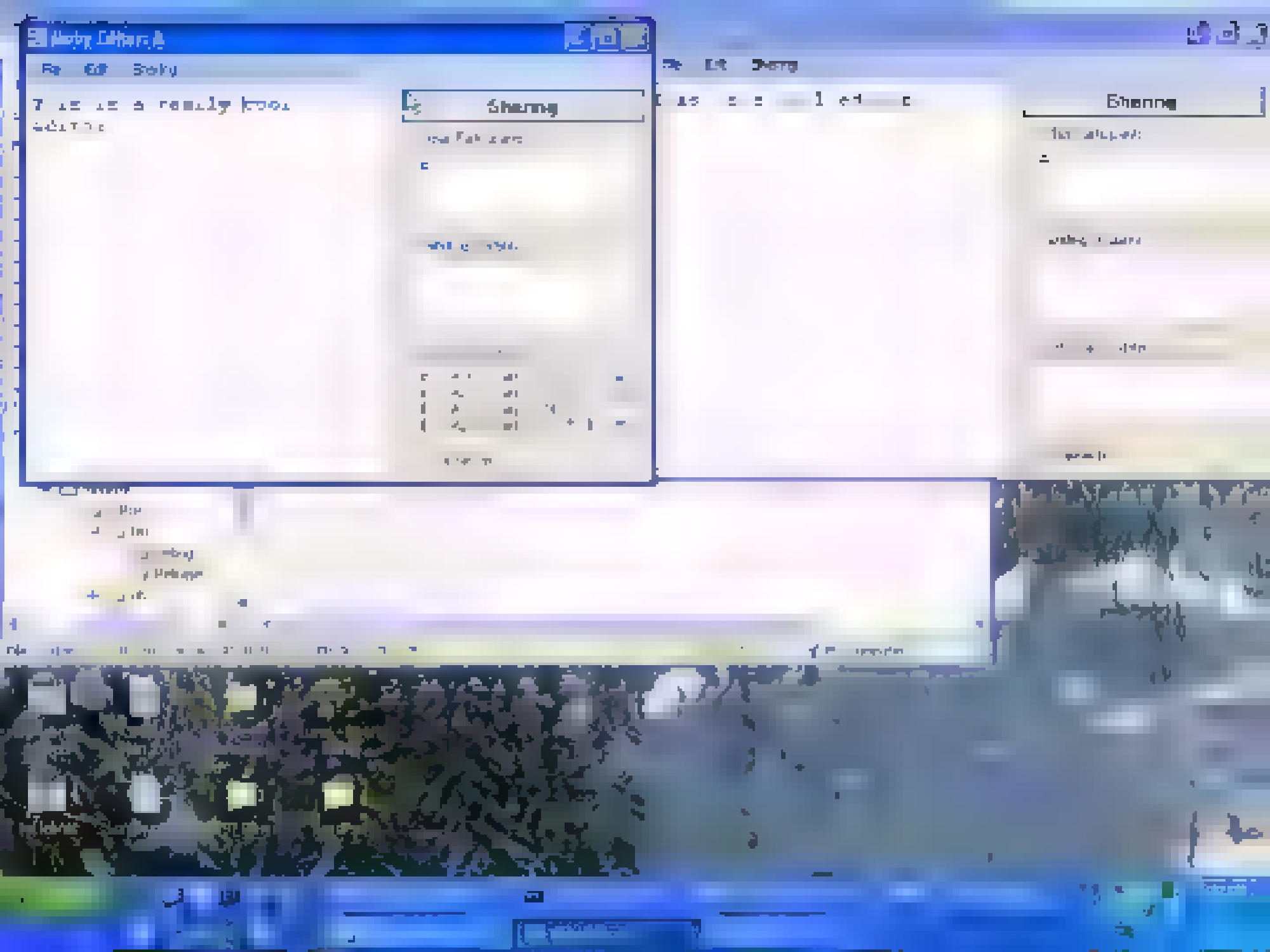
Defining correct transformations

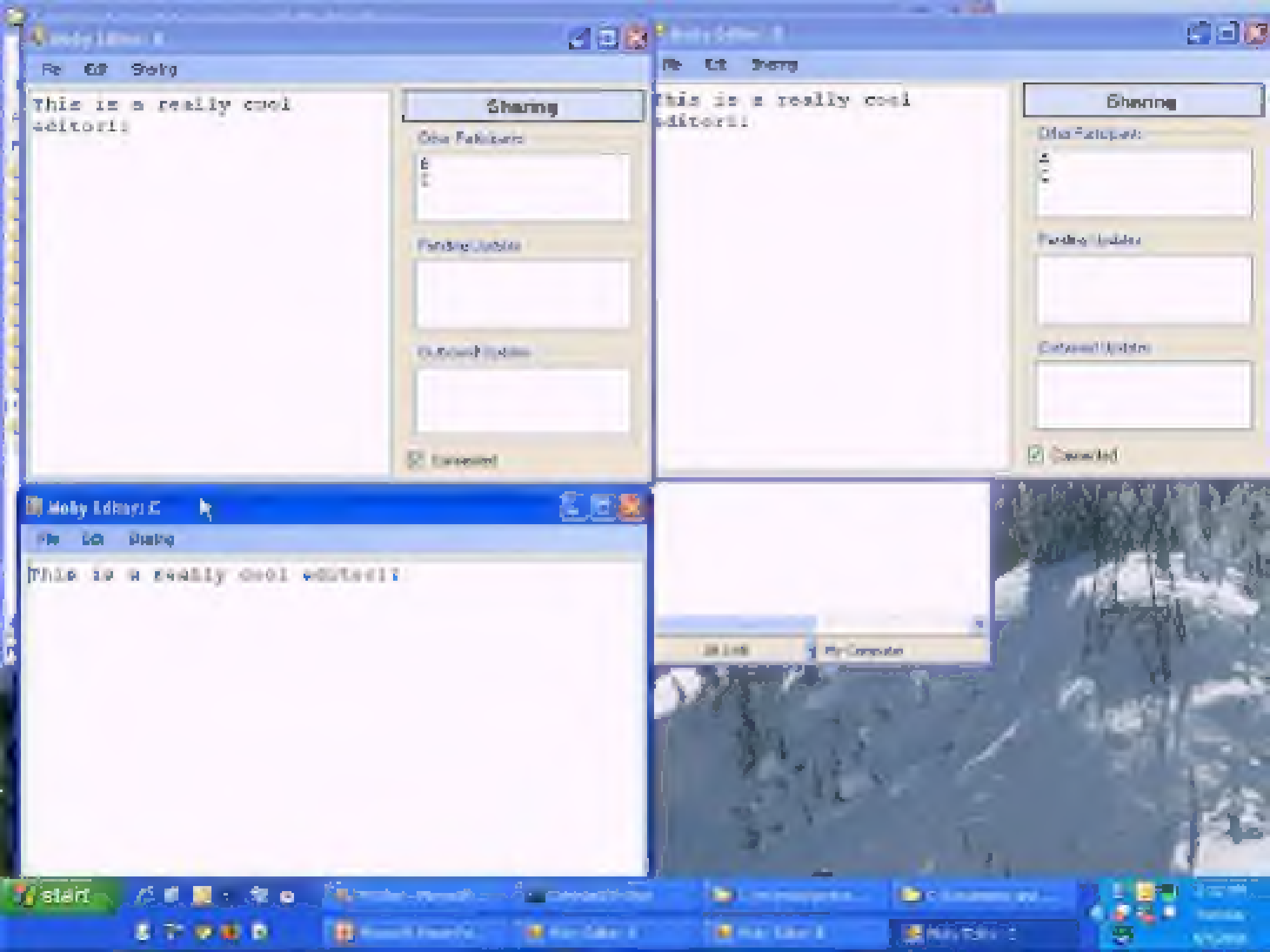
- Recall what "correctness" means:
  - Convergence
  - Intention preservation
- Invent a global ordering over all edits
  - E.g., lexicographic order of version vectors
- Simulate edits being applied in that order
  - (B; (A ^ B)) looks, intention-wise, like (A; B) or (B A)
  - All copies simulate the same order, so they converge
- Tricky case:
  - A comes earlier in global order than B ..
  - . but we've already applied B when A arrives
  - Must transform A to simulate having done it before B

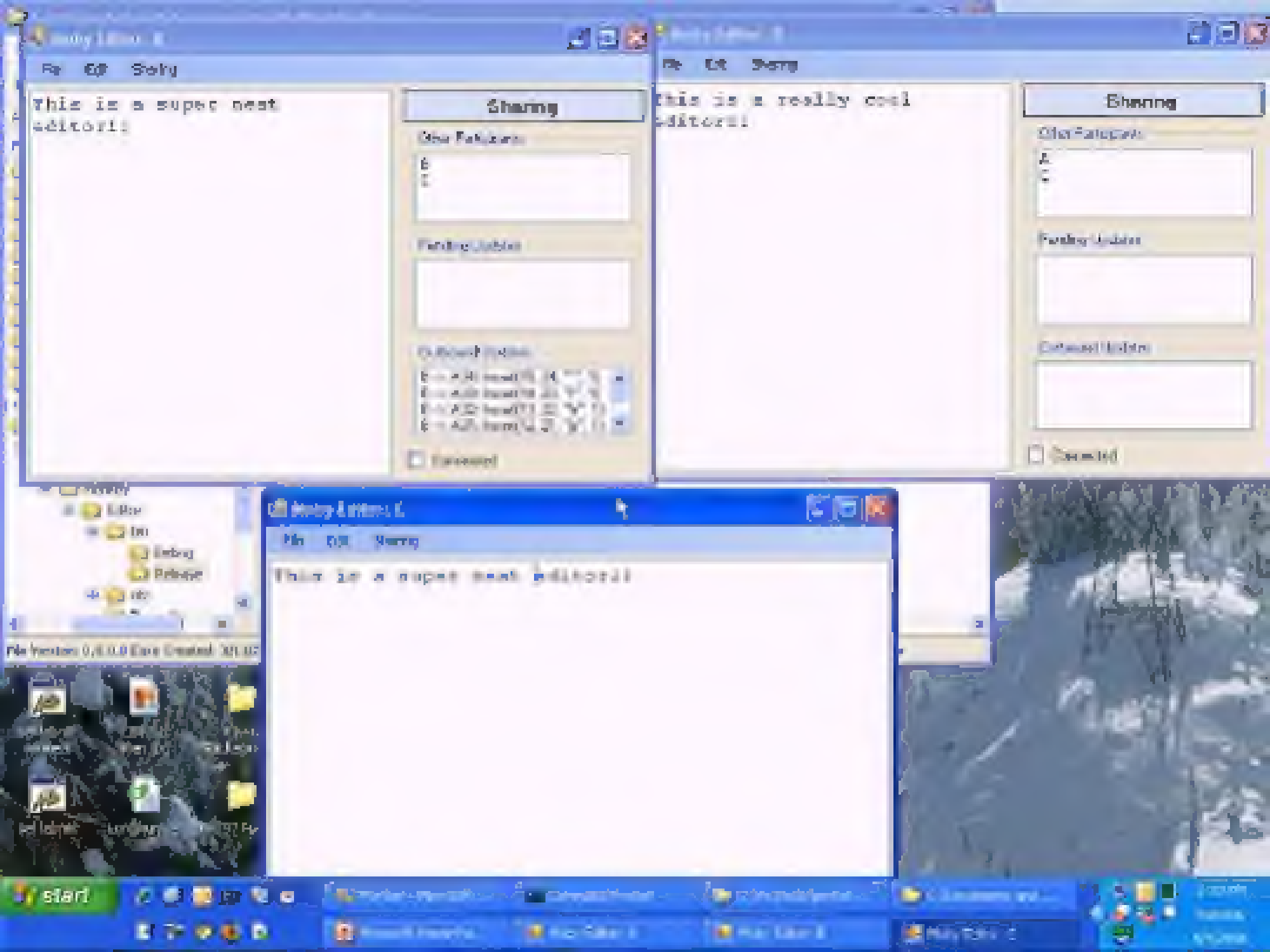
Demo

(This slide intentionally left blank)

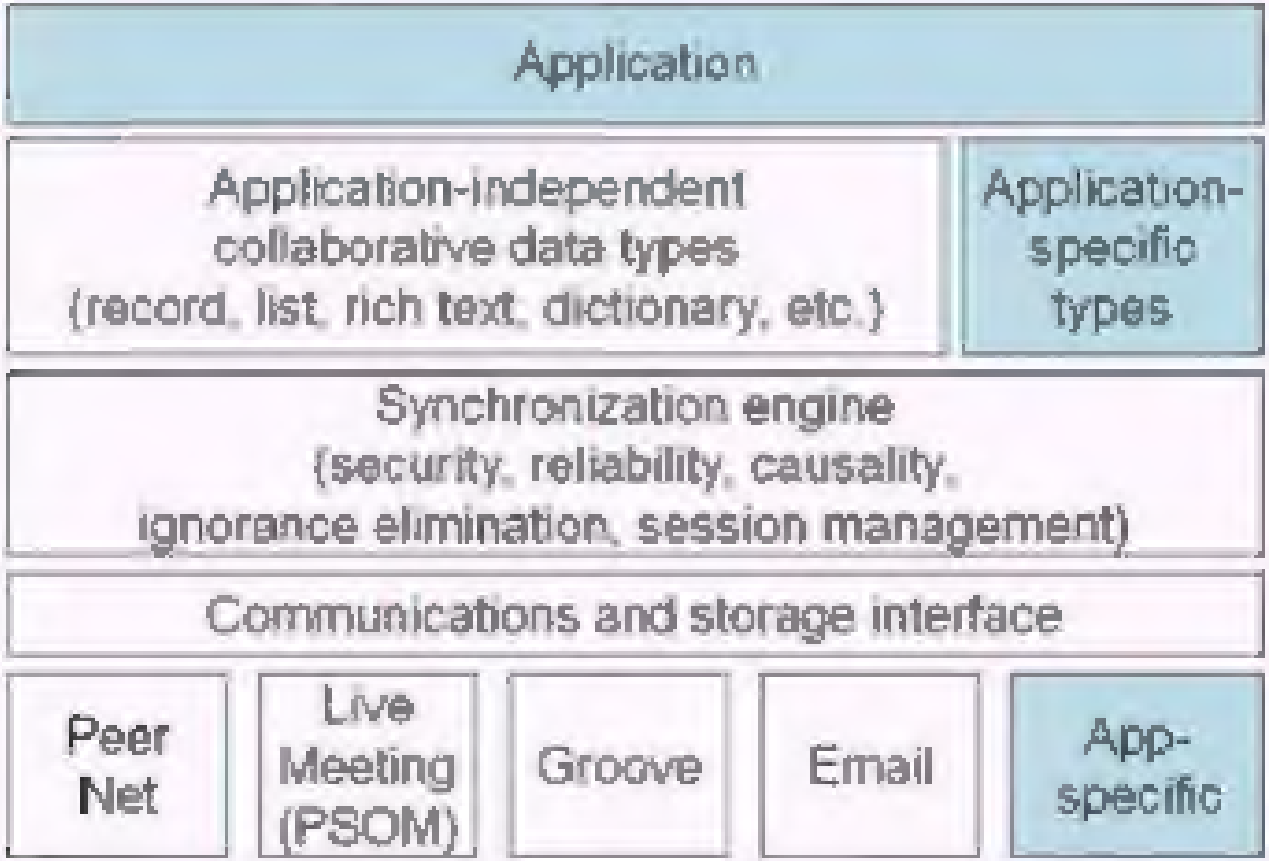





File Edit Sharing
This is a really cool editor!
Sharing
File Edit Sharing
This is a really cool editor!
Sharing
File Edit Sharing
This is a really cool editor!
My Computer
start


File Edit Sharing
This is a super neat editorial
Sharing
File Edit Sharing
This is a really cool editorial
Sharing
File Edit Sharing
This is a super neat editorial
start

Moby Architecture



Deliverables, ship vehicles, and the future

- Moby is a software library, not a product
  - Later, may release SDK to customers
- First Moby shipper: "Mojo" project in RTC
  - Release with Moby: early Spring 2007
- Who's next?
  - RTC's other products
  - One application in Office 14?
  - Many other desktop apps around Microsoft…